PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Trabalho Prático 1*

Model - Trabalho prático TGC - ICEI - PUC Minas

Andrei Gonçalves R. Massaini e Luiz Fernando C. Rodrigues[1]

Resumo

O trabalho prático envolve a identificação de blocos em um grafo biconexo, que é um grafo conexo sem articulações, ou seja, um grafo "tolerante a falhas". Um grafo é biconexo se e somente se cada par de seus vértices estiver ligado por dois caminhos internamente disjuntos. Os componentes biconexos do grafo são subgrafos maximais de G que sejam biconexos em vértices. Cada componente biconexo é também chamada bloco do grafo.

O trabalho prático propõe três métodos para identificação de blocos em um grafo biconexo: (i) um método que verifica a existência de dois caminhos internamente disjuntos (ou um ciclo) entre cada par de vértices do bloco; (ii) um método que identifica articulações testando a conectividade após a remoção de cada vértice; e (iii) o método proposto por Tarjan (1972).

O experimento consiste em avaliar o tempo médio gasto para as três estratégias aplicadas a grafos aleatórios contendo 100, 1.000, 10.000 e 100.000 vértices.

Palavras-chave: Template. LaTeX. Abakos. Periódicos.

*Artigo apresentado à Revista Abakos

[1]Bacharel em Ciência da computação PUC Minas, Brasil– andrei.massaini@hotmail.com

1 INTRODUÇÃO

Este trabalho tem como objetivo a identificação de blocos em um grafo. Blocos são subgrafos conexos que não podem ser decompostos em subgrafos menores. A identificação desses blocos é importante em diversas áreas, como em redes sociais, análise de circuitos elétricos, análise de comunidades em redes complexas, entre outros. Para a identificação dos blocos, utilizamos algoritmos específicos que exploram a estrutura do grafo e suas propriedades. Nesse trabalho, apresentaremos a implementação desses algoritmos e a análise dos resultados obtidos em grafos de diferentes tamanhos e topologias. Além disso, discutiremos as implicações desses resultados em aplicações práticas e possíveis direções futuras de pesquisa.

2 IMPLEMENTAÇÃO

No desenvolvimento do trabalho, utilizamos a linguagem java, por ser uma linguagem amplamente adotada, e bastante versátil. Inicialmente, partimos da representação de um grafo por uma matriz de adjacência, porém a implementação de um grafo por meio de uma lista de adjacência foi necessária, uma vez que não conseguimos gerar o grafo de 100.000 vértices pois houve estouro de memória.

2.1 Grafos implementados

Na pasta 'grafos Implementados', existem duas classes que são utilizadas para representar os grafos. Uma por meio de uma matriz de adjacência, e outra por uma lista de adjacência. Ambas possuem os métodos básicos de instanciar baseado no tamanho de vértices, assim como os métodos de adicionar arestas passando os seus respectivos vértices como parâmetro.

2.1.1 Algoritmos

Os algoritmos foram divididos em duas pastas: Na pasta 'ListMethods' armazena todos os métodos que foram implementados usando a lista de adjacência, já a pasta 'MatrizMethods' os métodos que foram implementados sobre a matriz de adjacência.

2.1.1.1 Caminhos Disjuntos - 1

O algoritmo implementa uma estratégia para identificar se um grafo é biconexo, ou seja, se cada par de vértices está conectado por dois caminhos internamente disjuntos. A ideia

principal é utilizar o algoritmo de Floyd-Warshall para calcular as distâncias entre todos os pares de vértices do grafo, e em seguida verificar se existem dois caminhos disjuntos entre cada par de vértices.

O algoritmo de Floyd-Warshall é utilizado para calcular as distâncias mais curtas entre todos os pares de vértices do grafo. Para isso, inicialmente, as matrizes de distância e proximidade são inicializadas com os valores do grafo de entrada. Em seguida, o algoritmo de Floyd-Warshall é executado, comparando as distâncias já calculadas com a possibilidade de obter uma distância menor passando por um vértice intermediário. Se isso for possível, a matriz de distância é atualizada com a nova distância e a matriz de proximidade é atualizada com o novo vértice intermediário.

Após a execução do algoritmo de Floyd-Warshall, o algoritmo verifica se existem dois caminhos disjuntos entre cada par de vértices do grafo. Para isso, é percorrida a matriz de distância e para cada par de vértices, é verificado se há dois caminhos disjuntos passando por um vértice intermediário. Se existirem, o vértice intermediário é marcado como uma articulação.

Por fim, o algoritmo retorna um vetor booleano indicando se cada vértice é uma articulação ou não. Se um vértice for marcado como articulação, significa que existem pelo menos dois caminhos disjuntos que o conectam a outros vértices, o que indica que o grafo é biconexo.

2.1.1.2 Find Components - 2

O algoritmo consiste em encontrar os pontos de articulação em um grafo representado por uma matriz de adjacência. Para isso, removemos cada vértice do grafo, e contamos o número de componentes do mesmo. Se o número de componentes for maior do que 1, o vértice removido é uma articulação. O algoritmo é implementado por meio de três funções: getNumConnectedComponents que conta o número de componentes conectados usando uma busca em profundidade (DFS); removeVertex que remove um vértice da matriz de adjacência e retorna a nova matriz; e getArticulacoes que percorre todos os vértices da matriz de adjacência, remove um por vez e conta o número de componentes conectados para identificar os pontos de articulação. O resultado é armazenado em um vetor de inteiros, onde o valor 1 indica que o vértice correspondente é uma articulação e 0 caso contrário.

2.1.1.3 Tarjan - 3

O algoritmo de Tarjan tem como objetivo, encontrar as Strongly Connected Components (SCCs) em um grafo, As SCCs são subgrafos em que é possível chegar de um vértice a qualquer outro vértice desse subgrafo.

O código começa definindo algumas variáveis de instância, como a matriz de adjacência, o tamanho da matriz, os identificadores, os identificadores baixos, uma pilha, entre outros. Em seguida, ele implementa o método "calcularSCCs", que inicia o processo de busca pelas SCCs.

Este método chama o método "visitar", que é o núcleo do algoritmo de Tarjan.

O método "visitar"visita cada vértice do grafo, atribui um identificador e um identificador baixo ao vértice e empilha-o na pilha. Em seguida, percorre os vértices adjacentes e, se um vértice ainda não foi visitado, chama o método "visitar"novamente recursivamente. O método também atualiza os identificadores baixos com base nos vértices adjacentes visitados.

Se o vértice visitado for o início de uma SCC, o método cria uma lista de SCCs e a preenche com os vértices desempilhados da pilha até o vértice inicial. Isso é repetido até que todos os vértices tenham sido visitados e todas as SCCs tenham sido identificadas.

3 ANÁLISE DE RESULTADOS

Nesta seção o desempenho dos algorítimos serão analisados com base em tempo gasto para encontrar os blocos.

Ao executar o algorítimo para 100, 1.000, 10.000 e 100.000 vértices dos grafos, para cada um dos três métodos propostos, chegamos em resultados de tempo de execução diferentes para cada situação, variando o número de vértices e o método utilizado, sendo um por matriz de adjacência e outro por lista de ajacência.

Um código de marcação de tempo foi adicionado a cada método proposto acima com o propósito de comparar a eficiência doss mesmos. Ao utilizar matriz de adjacência, conseguimos realizar os testes apenas com grafos de até 1.000 vértices, ao tentar rodar o algorítimo para grafos de 10.000 vértices ou mais, não foi possível realizar a medição, pois deu TIME OUT. Ao utilizar a lista de adjacência, conseguimos realizar os testas com todos os grafos propostos, não houve TIME OUT e a velocidade de processamento foi muito mais rápida.

3.1 Experimentos

Os testes foram realizados em um notebook com processador i7 8700k OC 5.0Ghz, 16Gb RAM.

Tempo de execução dos algoritmos(ms)

Método	[V]	Matriz Adjacência	Lista Adjacência
i	100	8	7
ii	100	13	7
iii	100	1	1
i	1.000	1885	1136
ii	1.000	2841	119
iii	1.000	7	1
i	10.000	TIMEOUT	TIMEOUT
ii	10.000	TIMEOUT	TIMEOUT
iii	10.000	TIMEOUT	TIMEOUT
i	100.000	TIMEOUT	TIMEOUT
ii	100.000	TIMEOUT	TIMEOUT
iii	100.000	TIMEOUT	TIMEOUT

Fonte: Dados da pesquisa

4 CONCLUSÃO

Após a realização de 90 testes com a Matriz de Adjacência, sendo 10 para cada método, e 90 testes com a Lista de adjacência, também sendo 10 para cada método, podem ser verificadas diversas diferenças e observações.

Apesar de ambos os algorítimos serem capazes de realizar os três métodos, pode-se perceber que para um grafo de pequeno porte, de aproximadamente 100 vértices, os dois são capazes de realizar, porém o algorítimo que utiliza a Lista de adjacência tem uma eficiência um pocuo melhor que o que utiliza Matriz de adjacência apenas para o segundo método, de aproximadamente 50 por cento, pois para o primeiro método e para o terceiro não houve quase nenhuma diferença. Ao evoluirmos os grafos para aproximadamente de 1.000 vértices, ambos os algorítimos foram eficientes, porém a utilização de Lista de adjacência foi aproximadamente 65 por cento mais rápido no primeiro método, 96 por cento mais rápida no segundo método e 84 por cento mais rapida no terceiro método.

Contudo conseguimos perceber que a melhor maneira de encontrar os blocos, através dos três métodos utilizados, é com a implementação de uma Lista de Adjacência, e utilizando o terceiro método proposto, o de método de Tarjan.